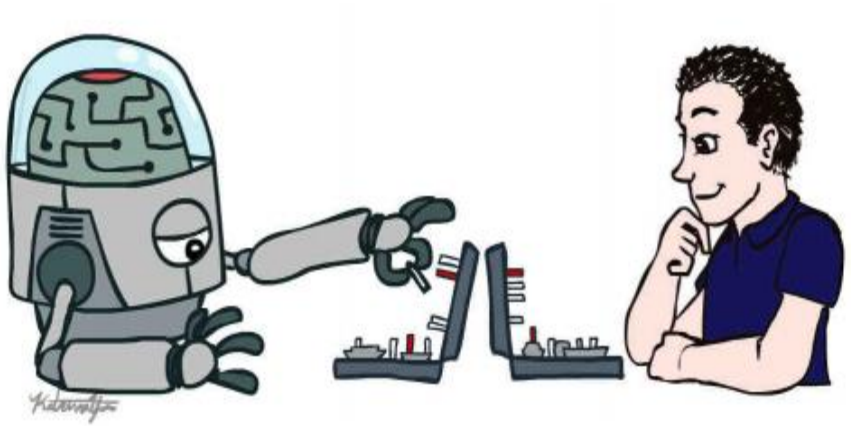
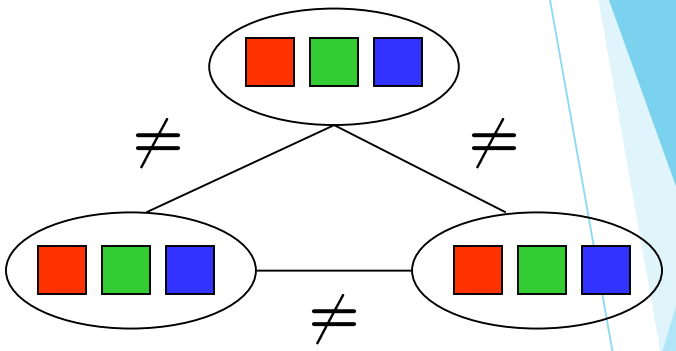
Artificial Intelligence
Search



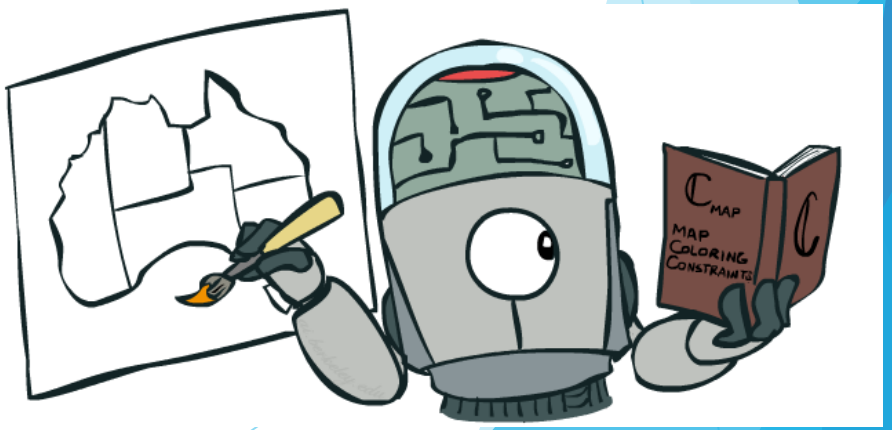
Dr Sherin ElGokhy

Reminder: CSPs

- CSPs:
  - Variables
  - Domains
  - Constraints
    - Implicit (provide code to compute)
    - Explicit (provide a list of the legal tuples)
    - Unary / Binary / N-ary



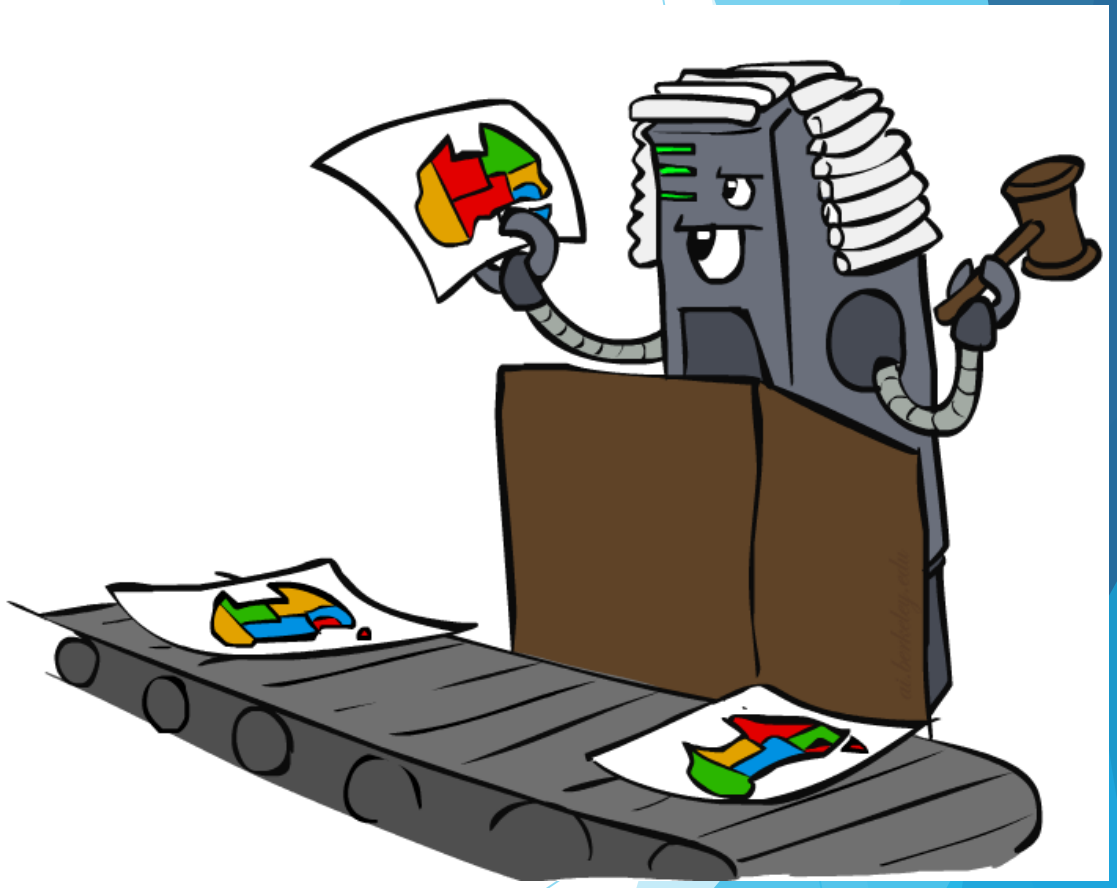
- Goals:
  - Here: find any solution
  - Also: find all, find best, etc.



Solving CSPs

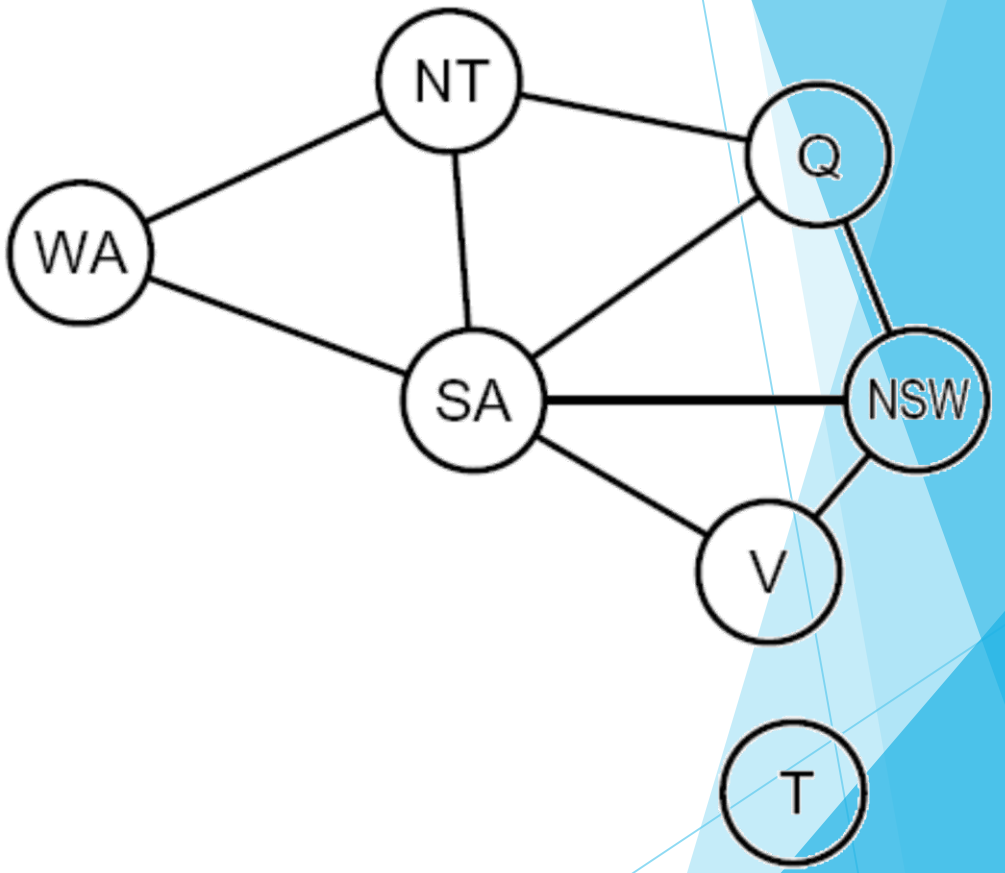
Standard Search Formulation

- Standard search formulation of CSPs
- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment, {}
  - Successor function: assign a value to an unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints
- We'll start with the straightforward, naïve approach, then improve it



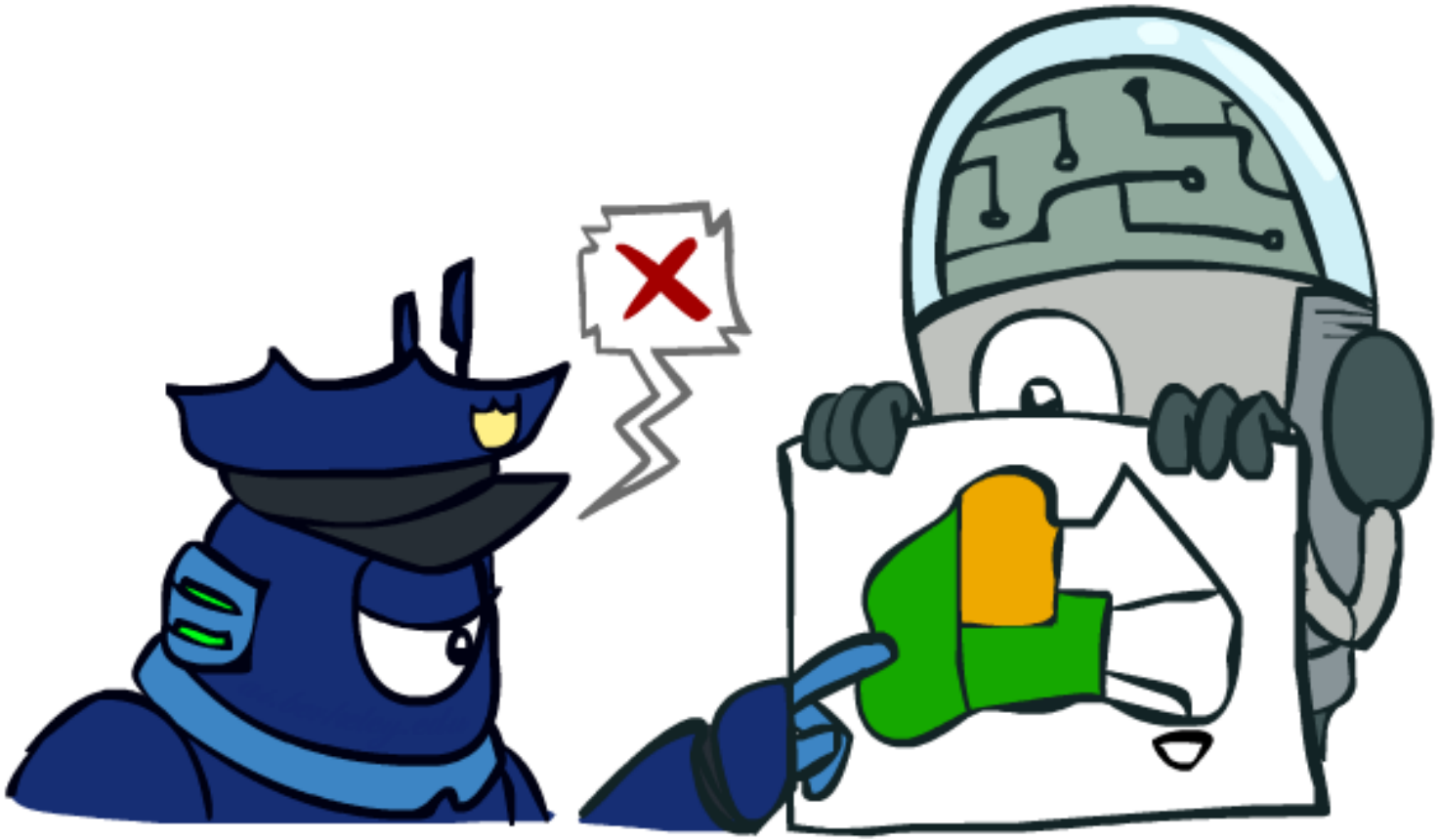
Search Methods

- What would BFS do?
- What would DFS do?



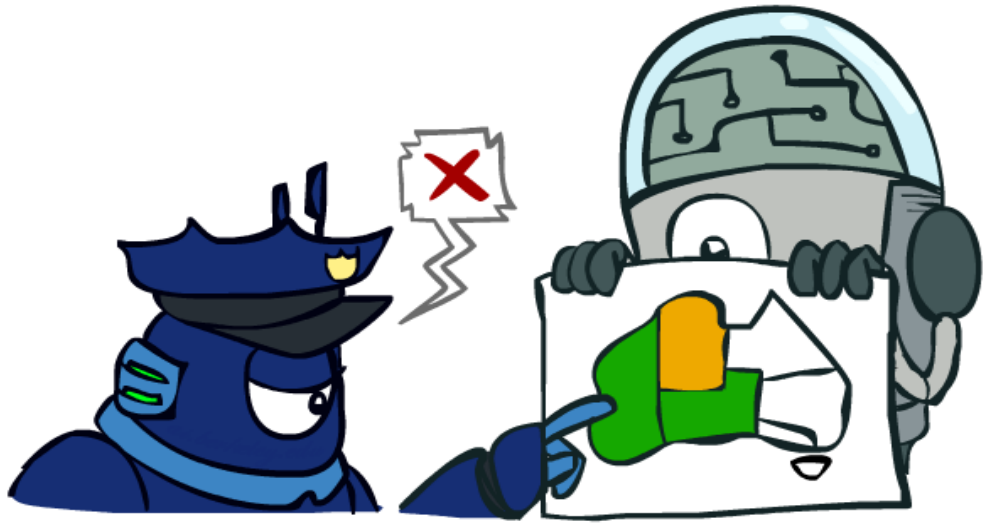
[Demo: coloring -- dfs]

Backtracking Search

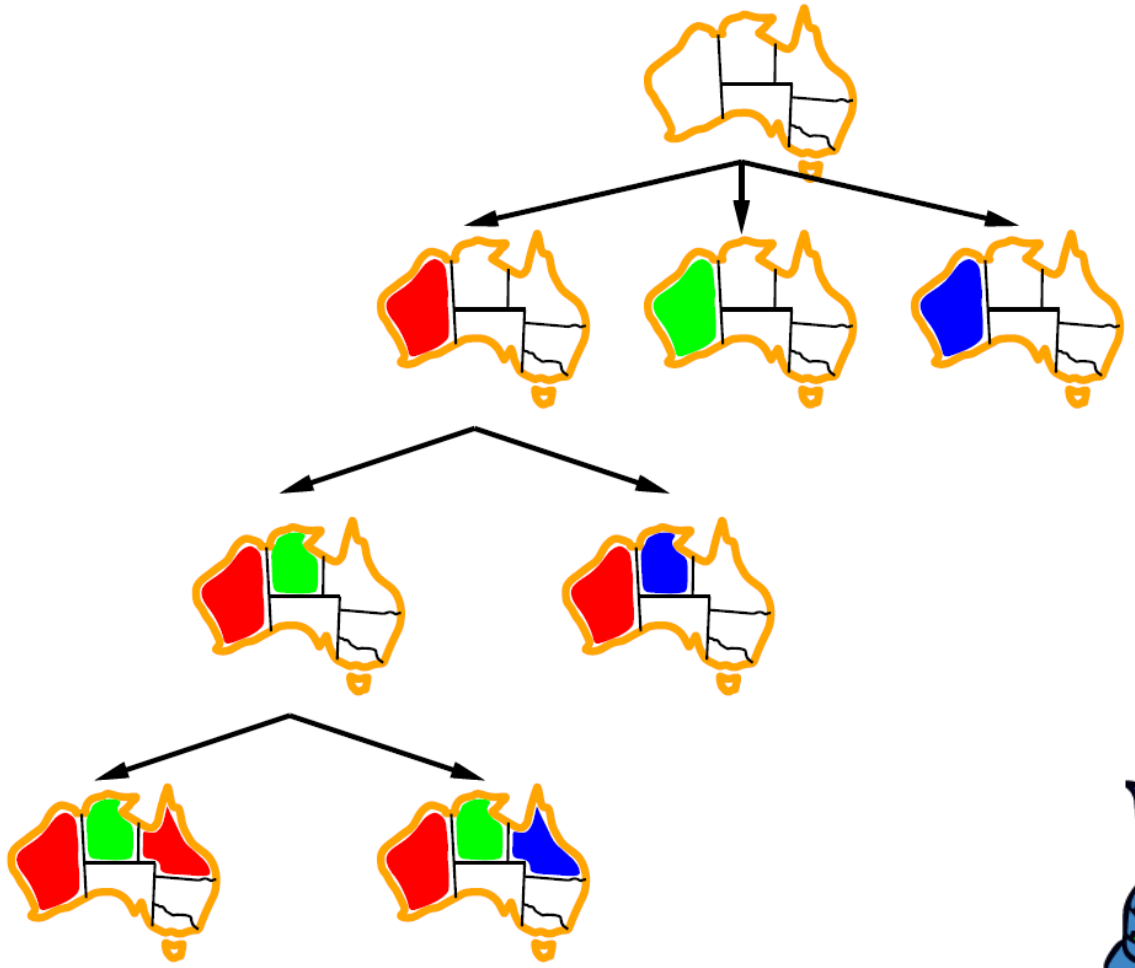


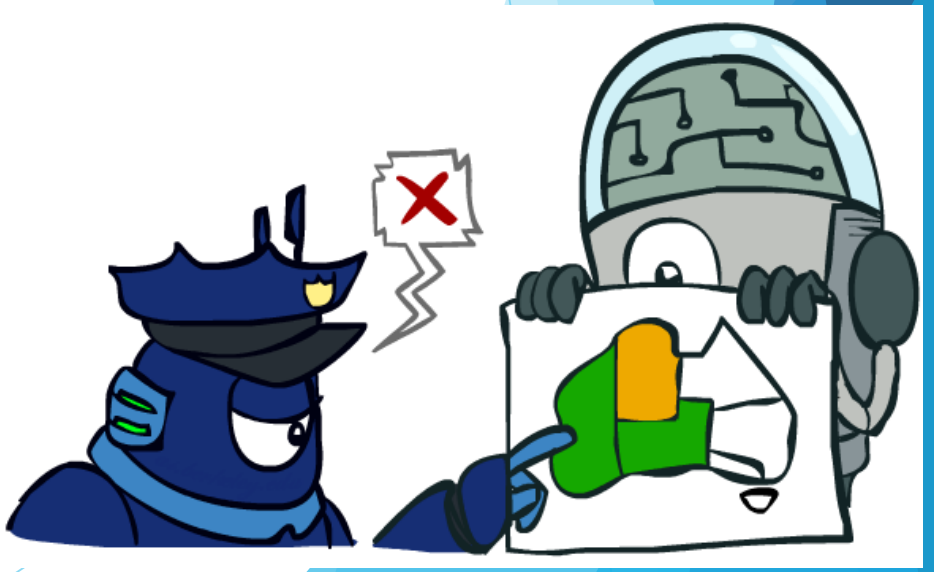
Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
- Idea 2: Check constraints as you go
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - "Incremental goal test"
- Depth-first search with these two improvements is called *backtracking search*
- Can solve n-queens for n ≈ 25



Backtracking Example





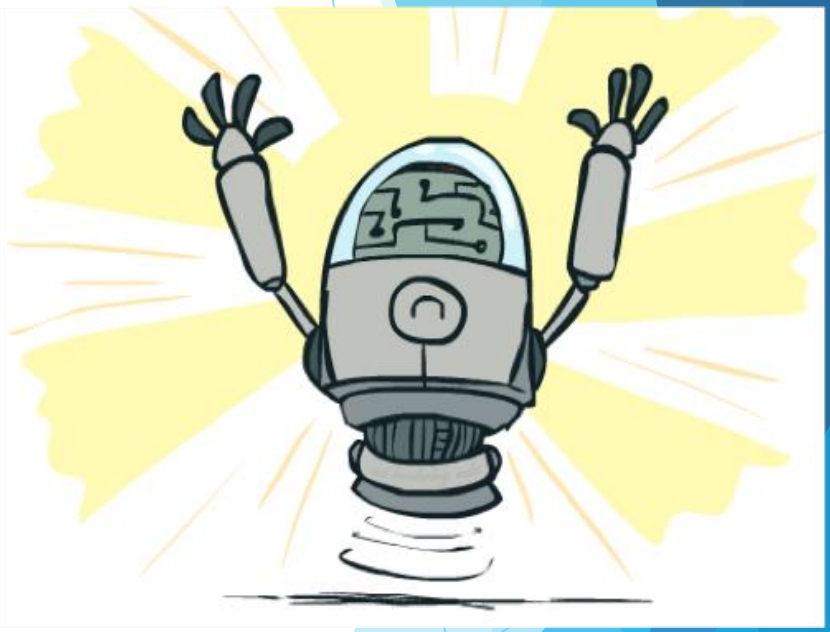
Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

- Backtracking = DFS + variable-ordering + fail-on-violation
- What are the choice points?

Improving Backtracking

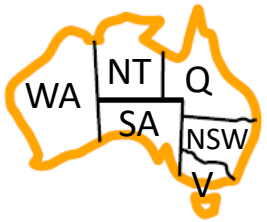
- General-purpose ideas give huge gains in speed
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering: Can we detect inevitable failure early?
- Structure: Can we exploit the problem structure?

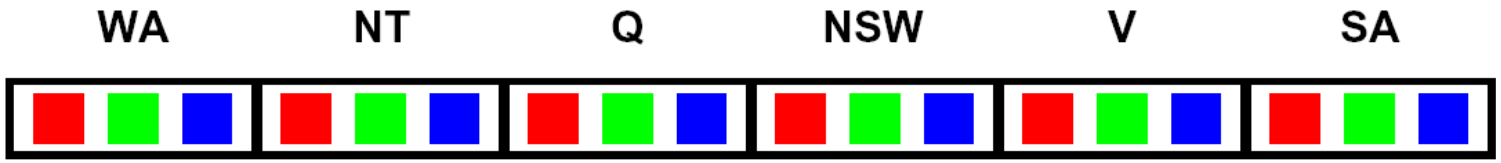


Filtering

Filtering: Forward Checking

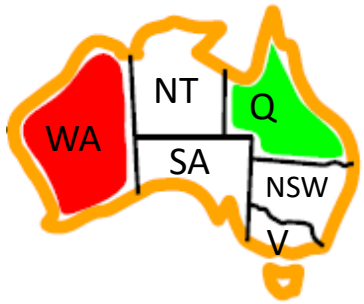
- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment

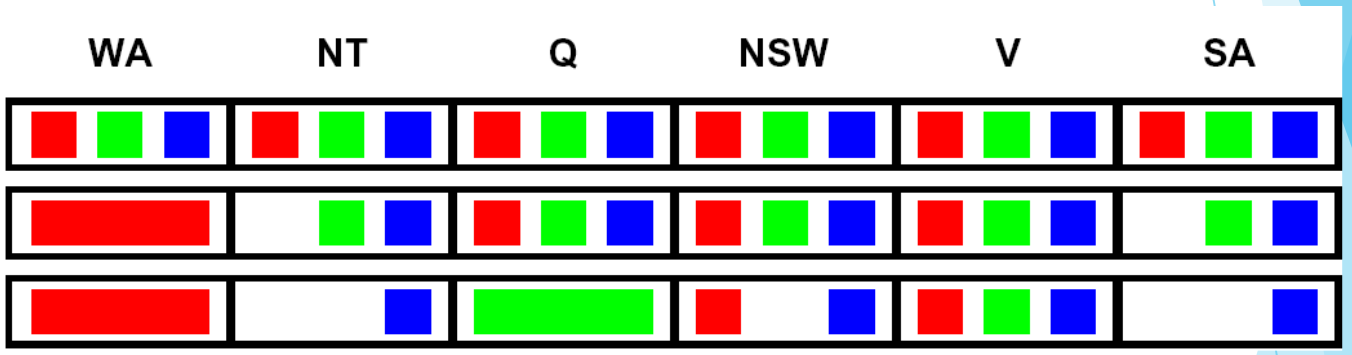




Filtering: Constraint Propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

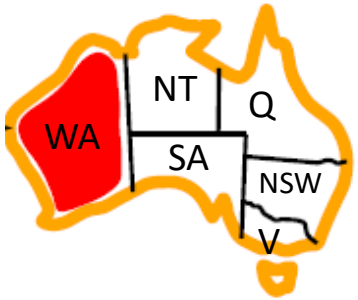


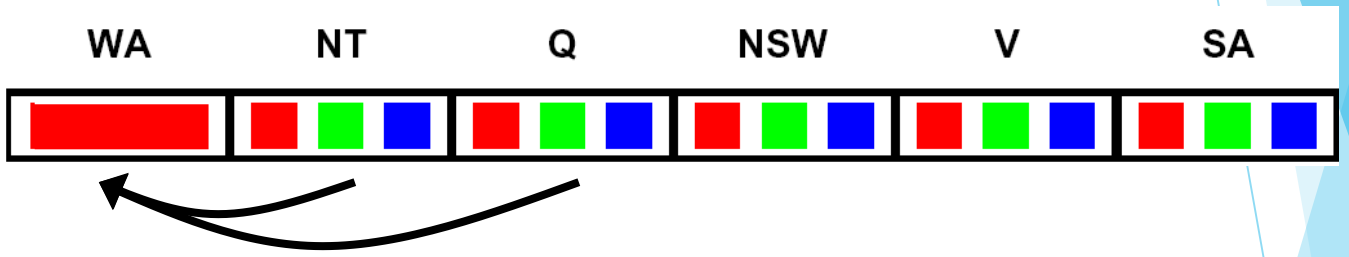


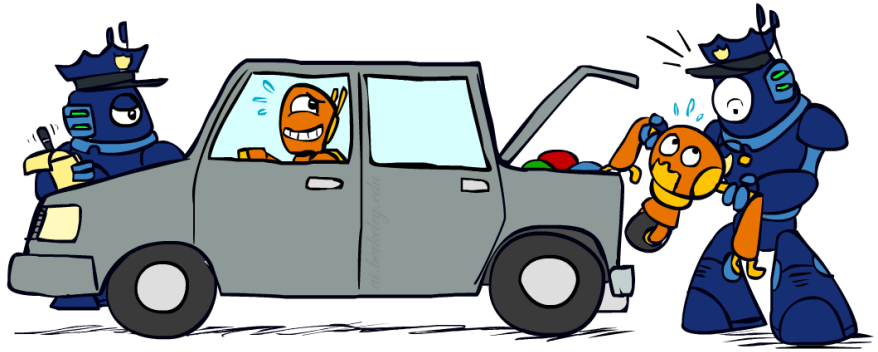
- NT and SA cannot both be blue!
- Why didn’t we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

Consistency of A Single Arc

- An arc X → Y is consistent iff for every x in the tail there is some y in the head which could be assigned without violating a constraint





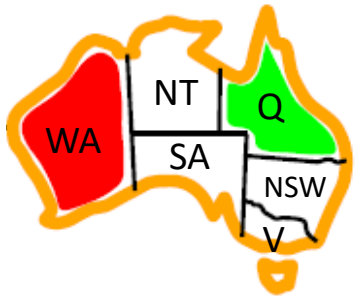


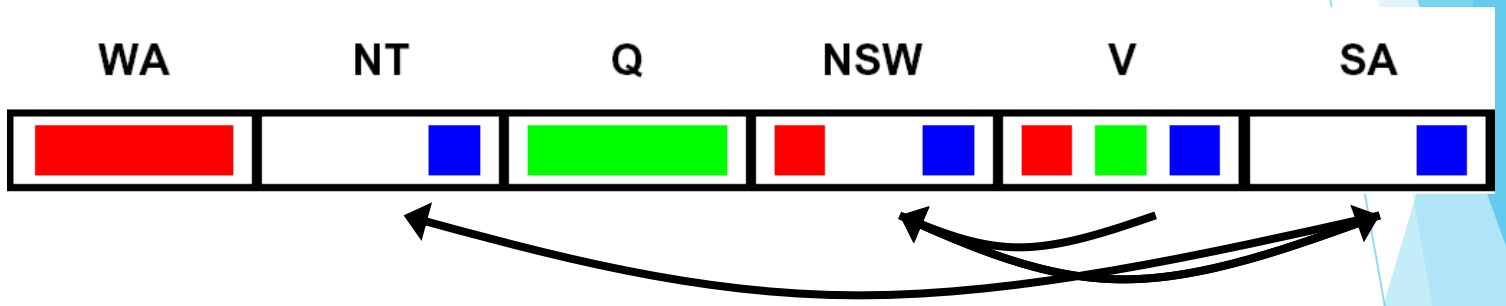
- Forward checking: Enforcing consistency of arcs pointing to each new assignment

Delete from the tail!

Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:





- Important: If X loses a value, neighbors of X need to be rechecked!
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment
- What's the downside of enforcing arc consistency?

Remember: Delete from the tail!

Enforcing Arc Consistency in a CSP

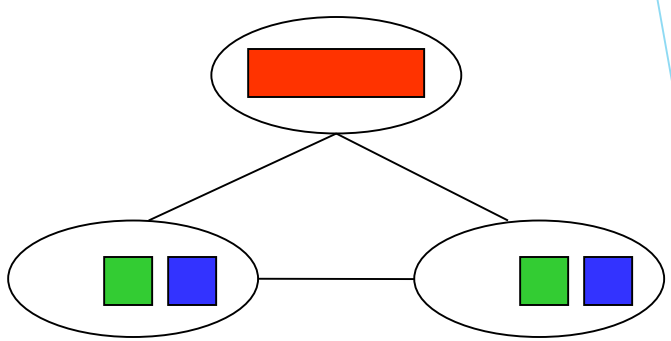
```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
      for each X_k in NEIGHBORS[X_i] do
        add (X_k, X_i) to queue

---------------------------------------------------------------------

function REMOVE-INCONSISTENT-VALUES( X_i, X_j) returns true iff succeeds
  removed ← false
  for each x in DOMAIN[X_i] do
    if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint X_i ↔ X_j
      then delete x from DOMAIN[X_i]; removed ← true
  return removed
```

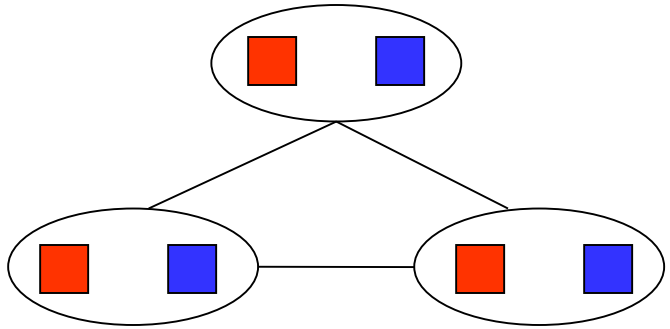
- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

Limitations of Arc Consistency

- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)

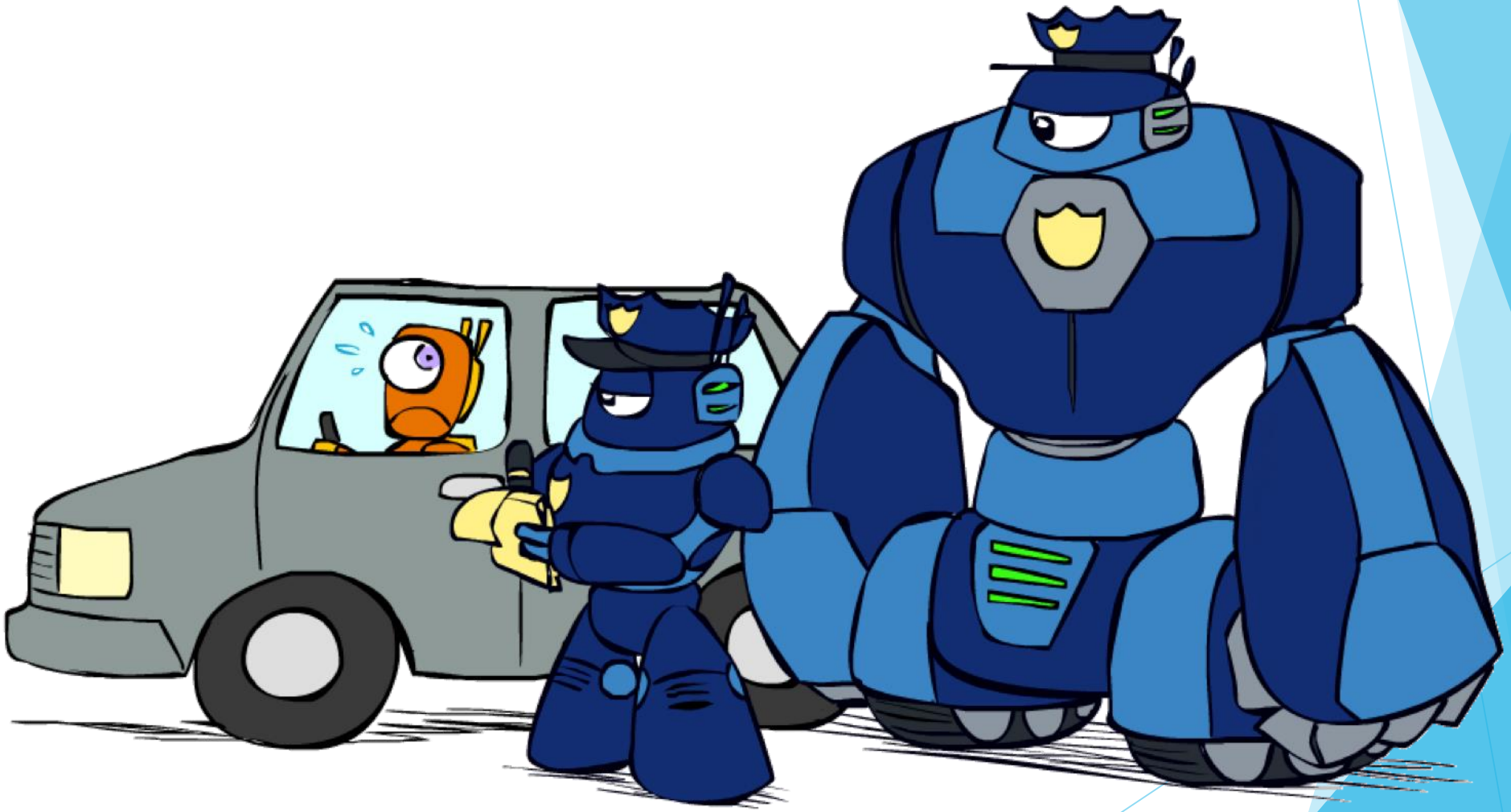
- Arc consistency still runs inside a backtracking search!





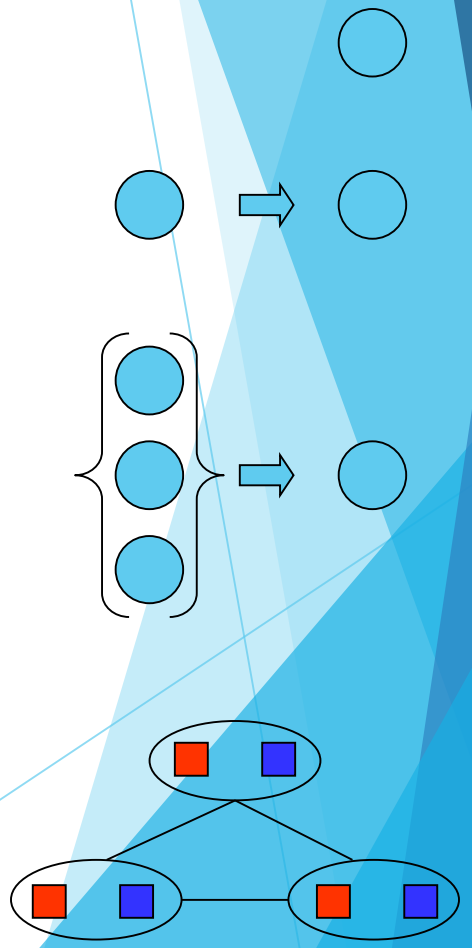
What went wrong here?

K-Consistency



K-Consistency

- Increasing degrees of consistency
  - 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints
  - 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other
  - K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the k^{th} node.
- Higher k more expensive to compute
- (You need to know the k=2 case: arc consistency)



Strong K-Consistency

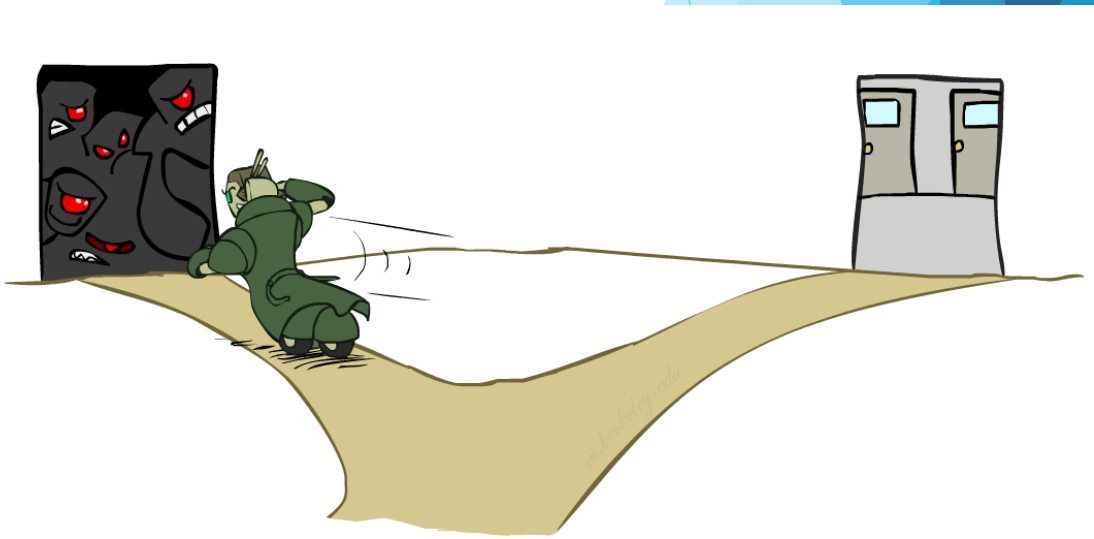
- Strong k-consistency: also k-1, k-2, … 1 consistent
- Claim: strong n-consistency means we can solve without backtracking!
- Why?
  - Choose any assignment to any variable
  - Choose a new variable
  - By 2-consistency, there is a choice consistent with the first
  - Choose a new variable
  - By 3-consistency, there is a choice consistent with the first 2
  - …
- Lots of middle ground between arc consistency and n-consistency! (e.g. k=3, called path consistency)

Ordering

Ordering: Minimum Remaining Values

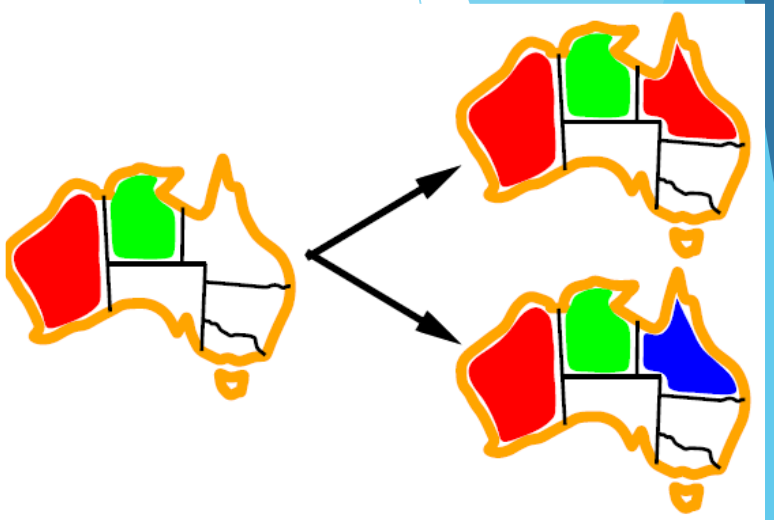
- Variable Ordering: Minimum remaining values (MRV):
  - Choose the variable with the fewest legal left values in its domain

- Why min rather than max?
- Also called “most constrained variable”
- “Fail-fast” ordering



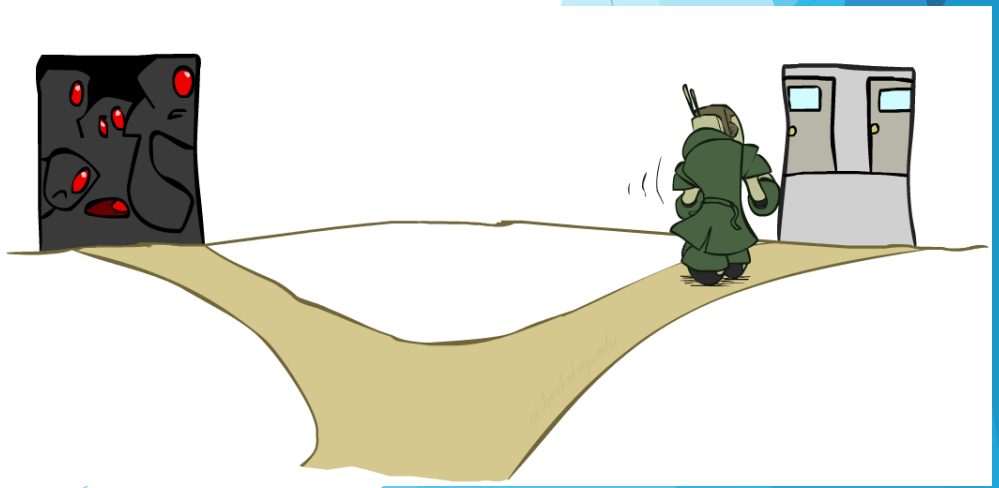
Ordering: Least Constraining Value

- Value Ordering: Least Constraining Value
  - Given a choice of variable, choose the *least constraining value*
  - I.e., the one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this! (E.g., rerunning filtering)



- Why least rather than most?

- Combining these ordering ideas makes 1000 queens feasible



Thanks